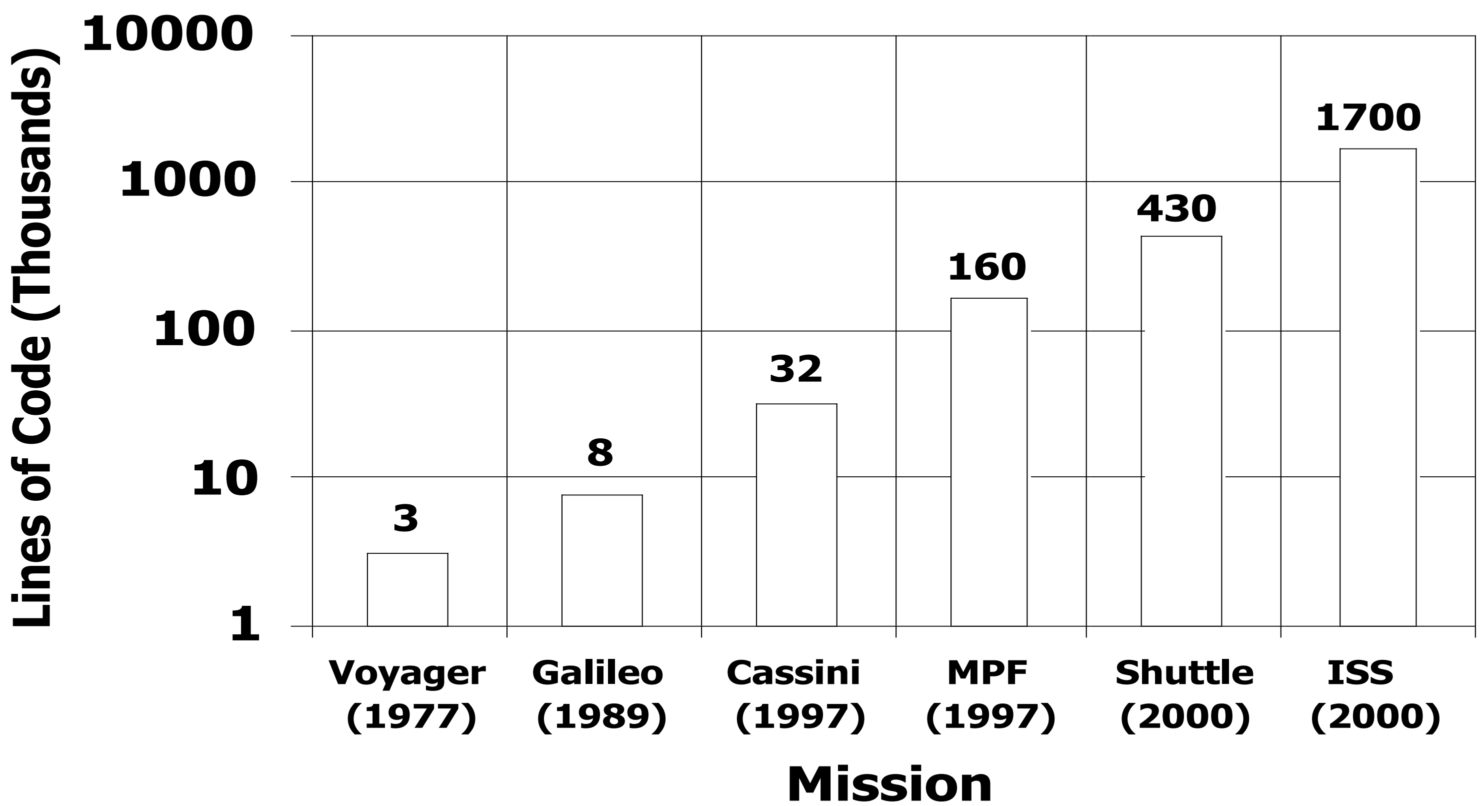
Experiences en analyse statique de logiciels embarqués

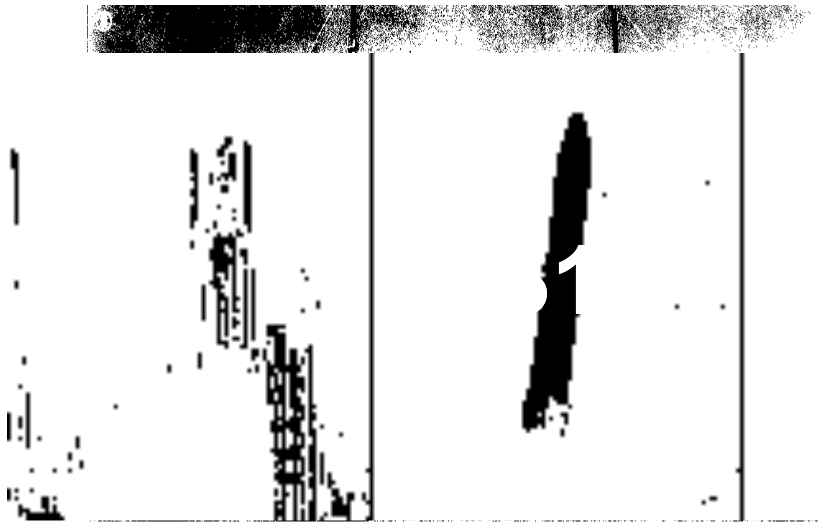
Experiences in the static analysis of embedded software

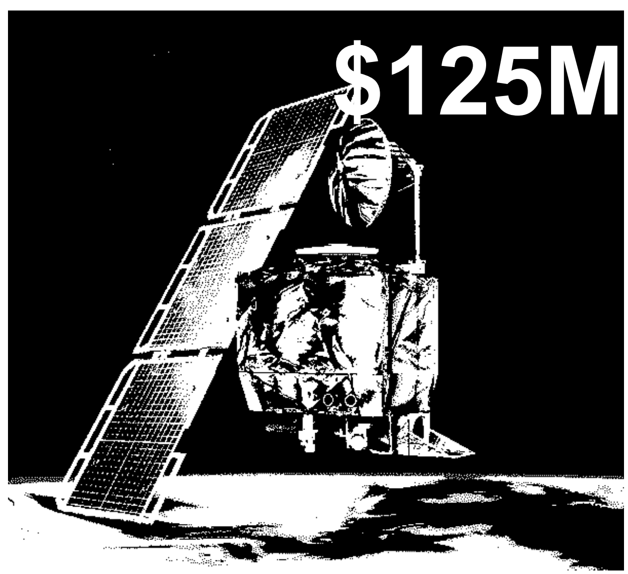
Guillaume Brat
(Kestrel Technology, Ames Division)

Software blowup



Famous aerospace failures

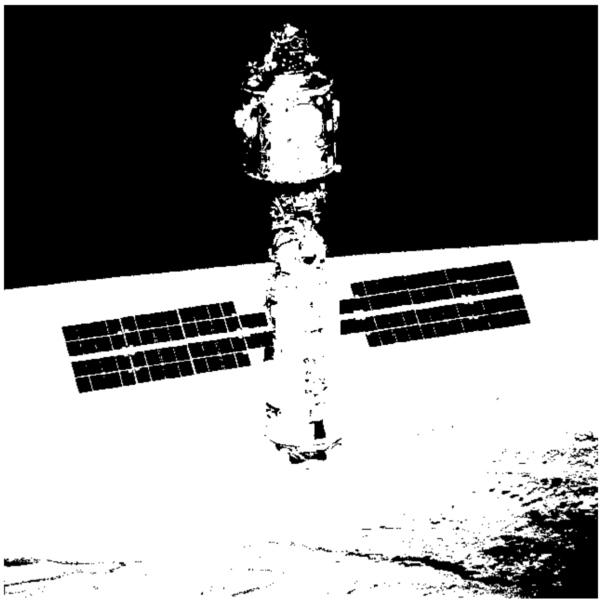




NASA Software Challenges

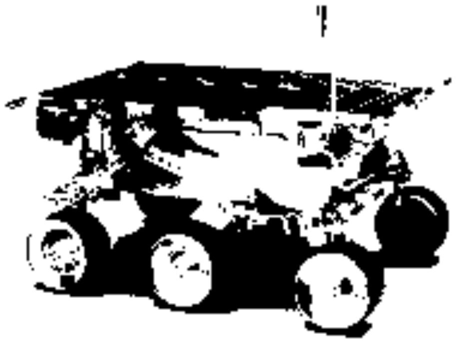
- Need to develop three systems for each mission:
  - Flight software
  - Ground software
  - Simulation software
- Flight software
  - Has to fit on radiation-hardened processors
  - Limited memory resources
  - Has to provide enough information for diagnosis
  - Can be patched (or uploaded) during the mission
- Each mission has its own goals, and therefore, each software system is unique!
- Cannot benefit from opening its source code to the public because of security reasons.
  - No open-source V&V
- Mission software is getting more complex.
  - Large source code (~1 MLOC)
  - The structure of the code is more complex

International Space Station

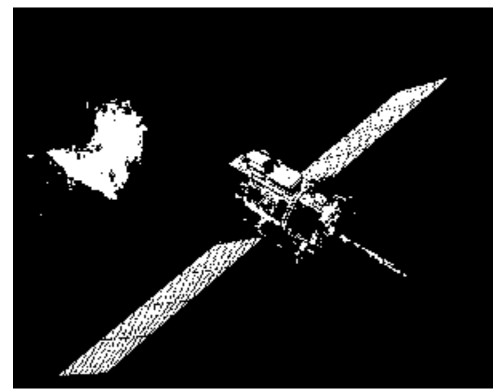


- International Space Station:
  - Attitude control system, 1553 bus, science payloads
  - International development (interface issues)
  - Codes ranging from 10-50 KLOC
  - A failure in a non critical system can cause a hazardous situation endangering the whole station
  - Enormous maintenance costs

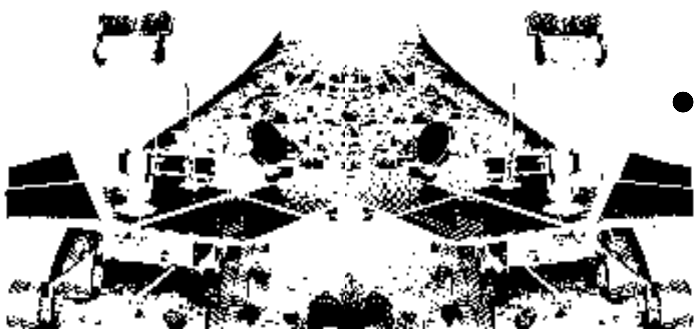
Mars mission software



- Mars Path Finder:
  - Code size: 140 KLOC
  - Famous bug: priority inversion problem



- Deep Space One:
  - Code size: 280 KLOC
  - Famous bug: race condition problem in the RAX software



- Mars Exploration Rovers:
  - Code size: > 650 KLOC
  - Famous bug: Flash memory

How is the Software Verified?

- Mars missions: high-fidelity test bench
  - Runs 24 hours a day
  - 8 hour test sessions:
- Space Station:
  - Critical software: on-ground simulator maintained at Marshall Space Center
  - Payloads:
    - Independently verified by contractors
    - NASA test requirement document

How effective is this?

- Badly re-initialized state variable for MPL:
- Unit mismatch for MCO:
- Thread priority inversion problem for MPF:
- Flash memory problem for MER:
- Science mission for the ISS currently under validation:
  - Passes NASA test requirements

Static Analysis

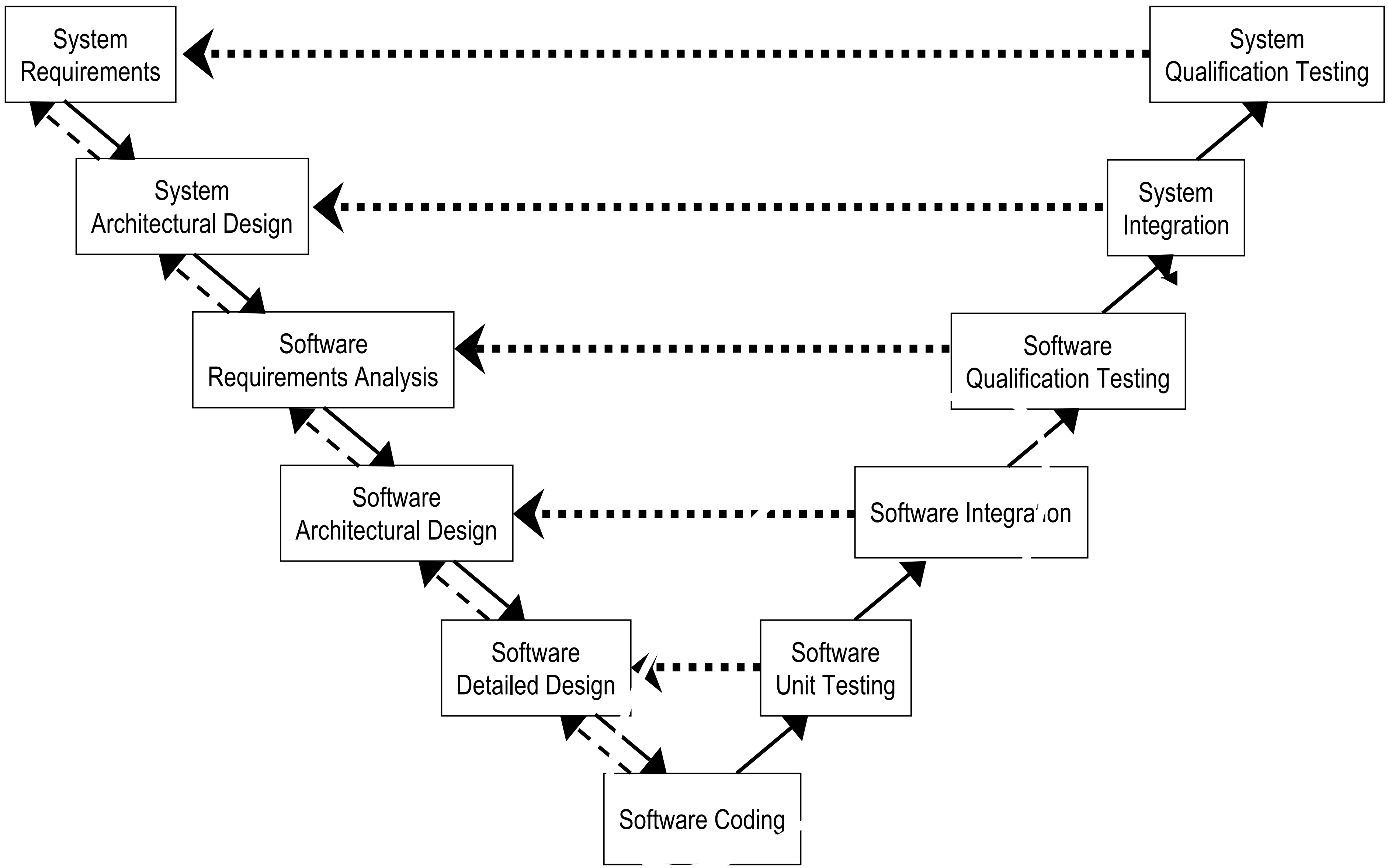
Static analysis offers compile-time techniques for predicting safe and computable approximations to the set of values arising dynamically at run-time when executing the program

We use abstract interpretation techniques
to extract a safe system of semantic equations
which can be resolved using lattice theory techniques
to obtain numerical invariants for each program point

Covered Defect Classes

- Static analysis is well-suited for catching runtime errors, e.g.:
  - Array-out-bound accesses
  - Un-initialized variables/pointers
  - Overflow/Underflow
  - Invalid arithmetic operations
- Defect classes for Deep Space One:
  - Misuse:
  - Initialization: , incorrect value
  - Assignment: wrong value,
  - Undefined Ops:
  - Omission:
  - Scoping Confusion: global/local, static/dynamic
  - Argument Mismatches:
  - Finiteness:

Software Development Process



Research Process

Our goal was to assess the capabilities of static analysis and identify the technical gaps to make it usable in NASA missions.

Identification of technical gaps

Identification of commercial tools

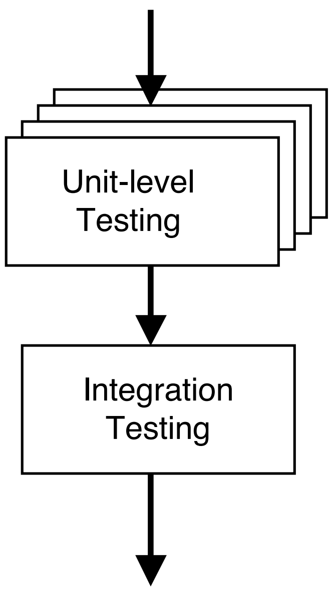
Experiments on real NASA code

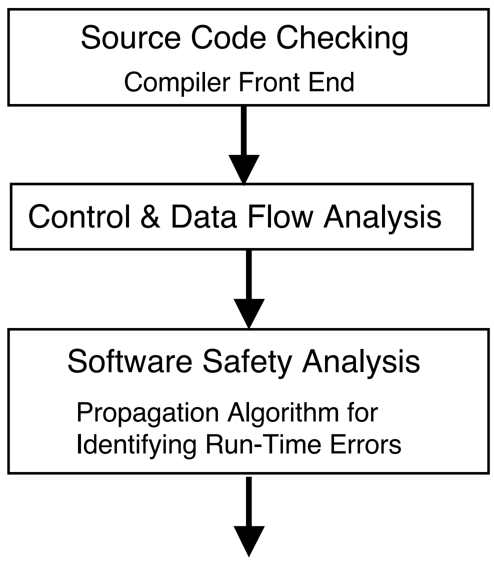
Implementation of research prototype

PolySpace C-Verifier

PolySpace C-Verifier finds runtime errors in C programs.

It works like a sophisticated compiler.

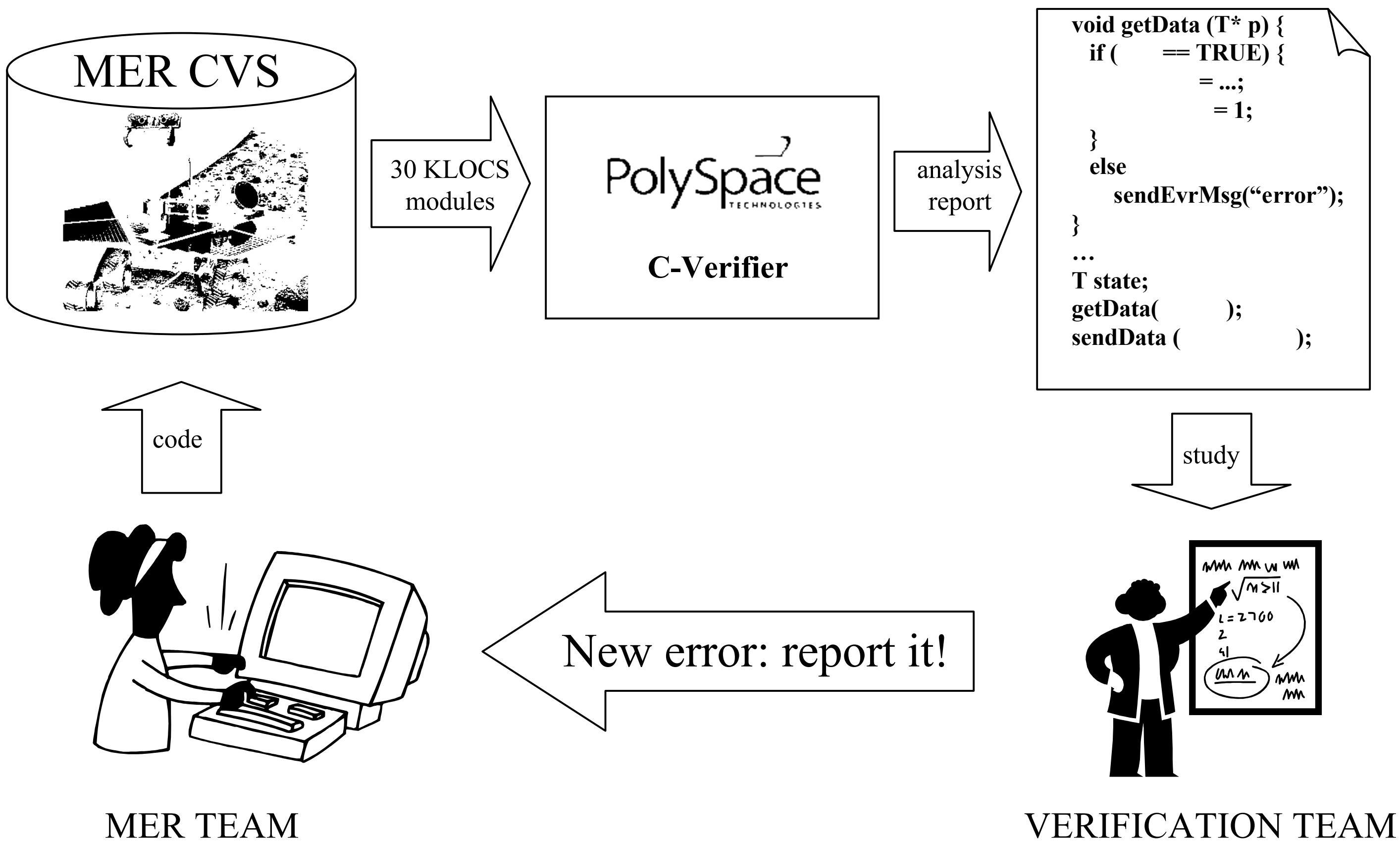





```
    p =    - 0.75;
    y =      ( );
}

/* unreachable or dead code
void unr () {
    int x = random_int );
    int y = random_int );
        if (  >  ) {
                x =      ;
                if (   < 0) {
```

color-coded reporting:
- always correct
- always incorrect
- may be incorrect
- never executed

STATIC ANALYSIS OF MER



Experimental results

Project	MPF	ISS	MER
Language	C	C	C
Size	200KLocs	40KLocs	650KLocs
Maturity	Stable	Untested	Under-development
Modules	ACS+EDL	HLRC	bc, reu, pyro, pwr, dat, adc, pas, imu, mcas, rpdu, bcp, btp, …
Max Size	25KLocs	17KLocs	3.2KLocs
Errors	NIV	OBAI OVFL	NIV

Performance

- Pyro + Pwr modules:
  - 1st pass: O1, 54 mn, 4610 green, 601 orange
  - 2nd pass: O1, 44 mn, 4758 green, 409 orange
  - 2nd pass: O2, 34 mn, 4758 green, 409 orange
  - No significant red (obvious infinite loops)
- Dat + (adc, pas, imu, mcas, rpdu, pwr, pyro, bcp, btp)
  - Quick analysis: 30 mn
  - Un-initialized variable (not yet fixed)
  - Returning the address of a local variable (already fixed)
  - Overflow in constant expression (already fixed)

A Role for Static Analysis

- Extensive experiments with PolySpace Verifier:
  - Minors bugs found in MER
  - Serious out-of-bounds array accesses found in an ISS Science Payload

- Useful:
- Effective:
  - It takes 24 hours to analyze 40 KLOC
  - Difficulty to break down large systems into small modules

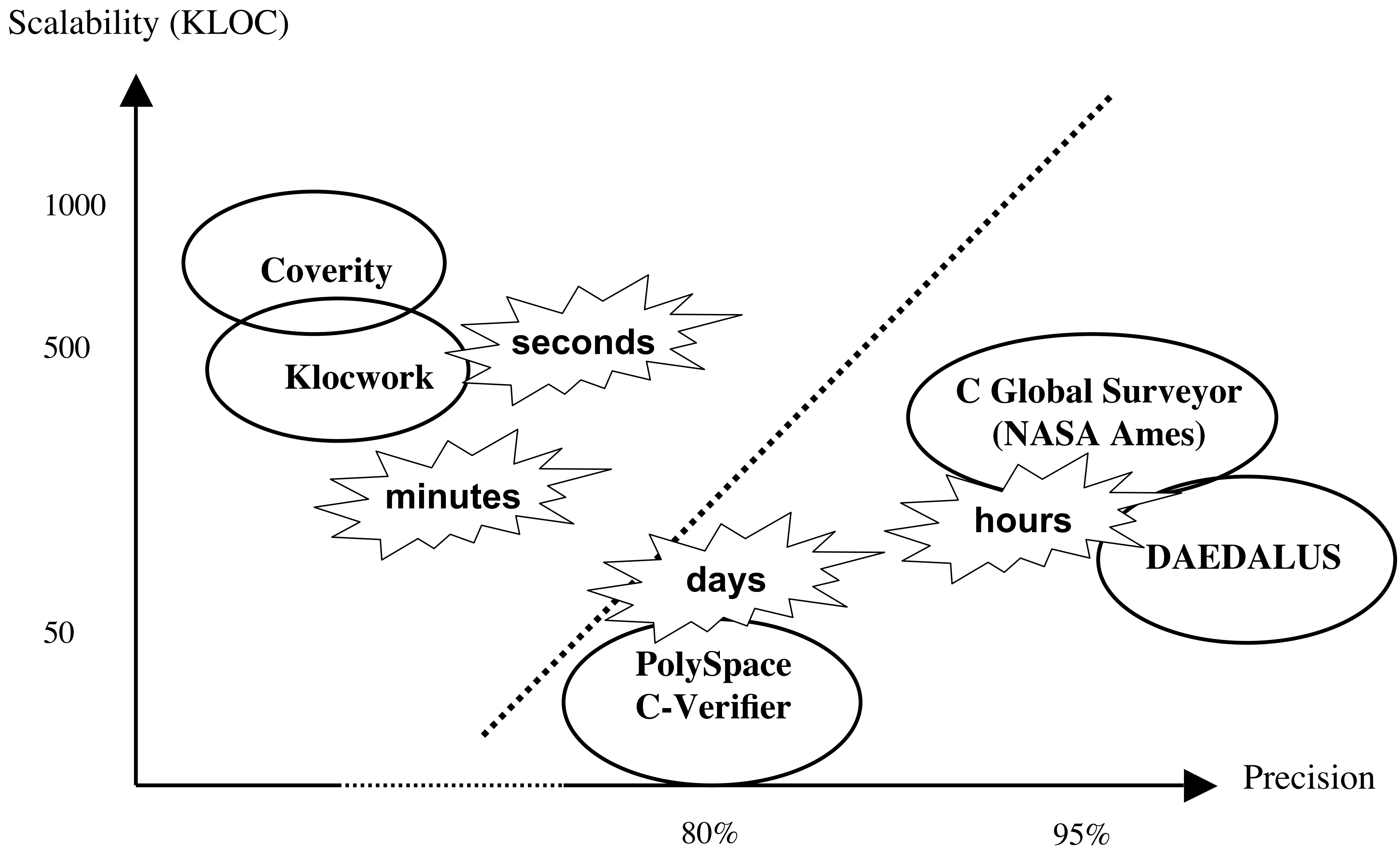
NASA Requirements

- Analyze large systems in less than 24 hours
- Analysis time similar to compilation time for mid-size programs

- Precision:
  - At least 80%

    the analysis provides enough information to diagnose a warning

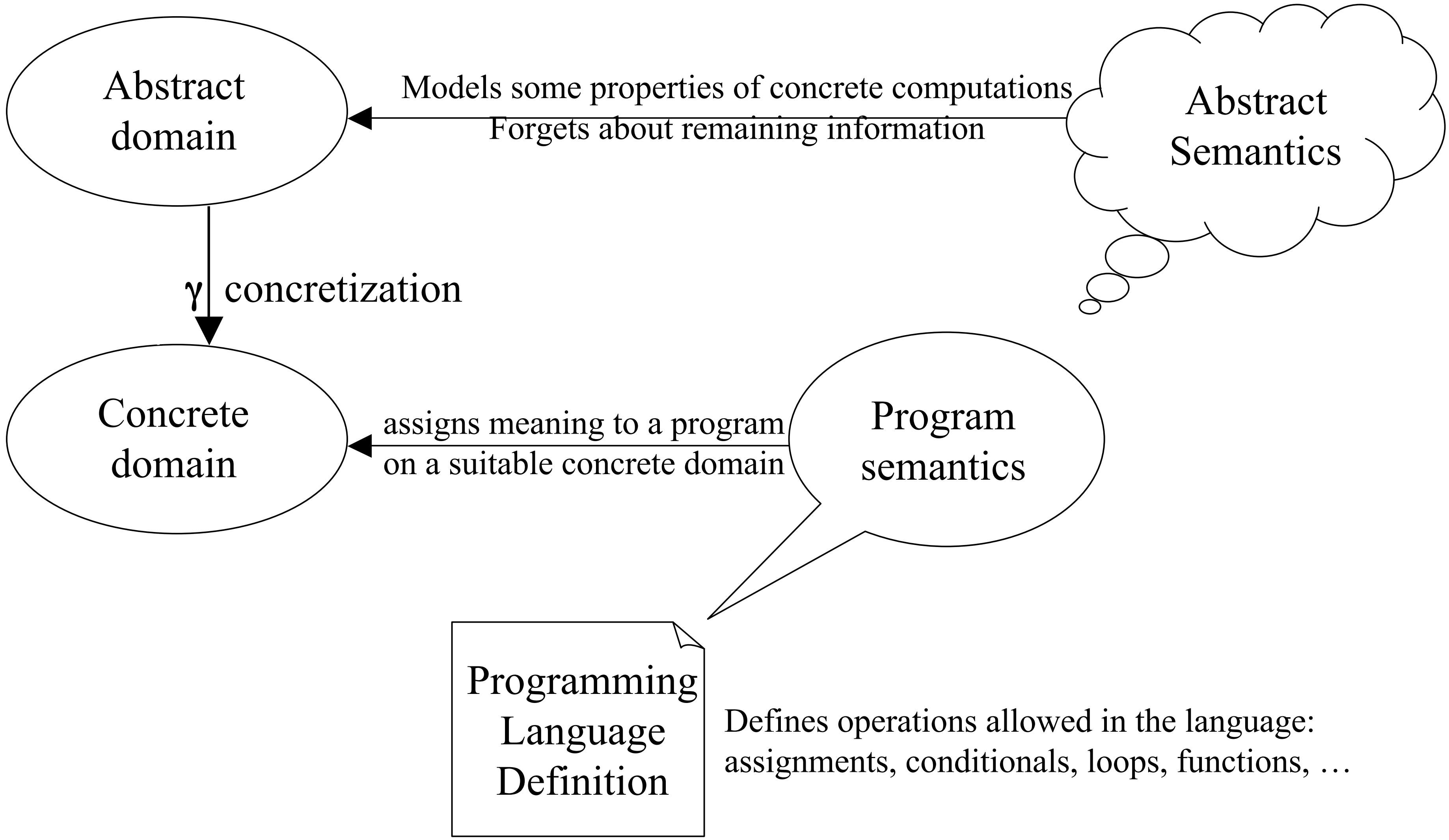
Practical Static Analysis



C Global Surveyor

- Prototype analyzer
  - Based on abstract interpretation
  - specialized for NASA flight software
- Covers major pointer manipulation errors:
  - Out-of-bounds array indexing
  - Un-initialized pointer access
  - Null pointer access
- Keeps all intermediate results of the analysis in a human readable form:

Abstract Interpretation



- Check that every operation of a program will never cause an error (division by zero, buffer overrun, deadlock, etc.)
- <u>Example:</u>

```
int a[1000];
for (i = 0; i < 1000; i++) {
       = … ;  // 0 <= i <= 999
}
     = … ;    // i = 1000;
```

Simple Example

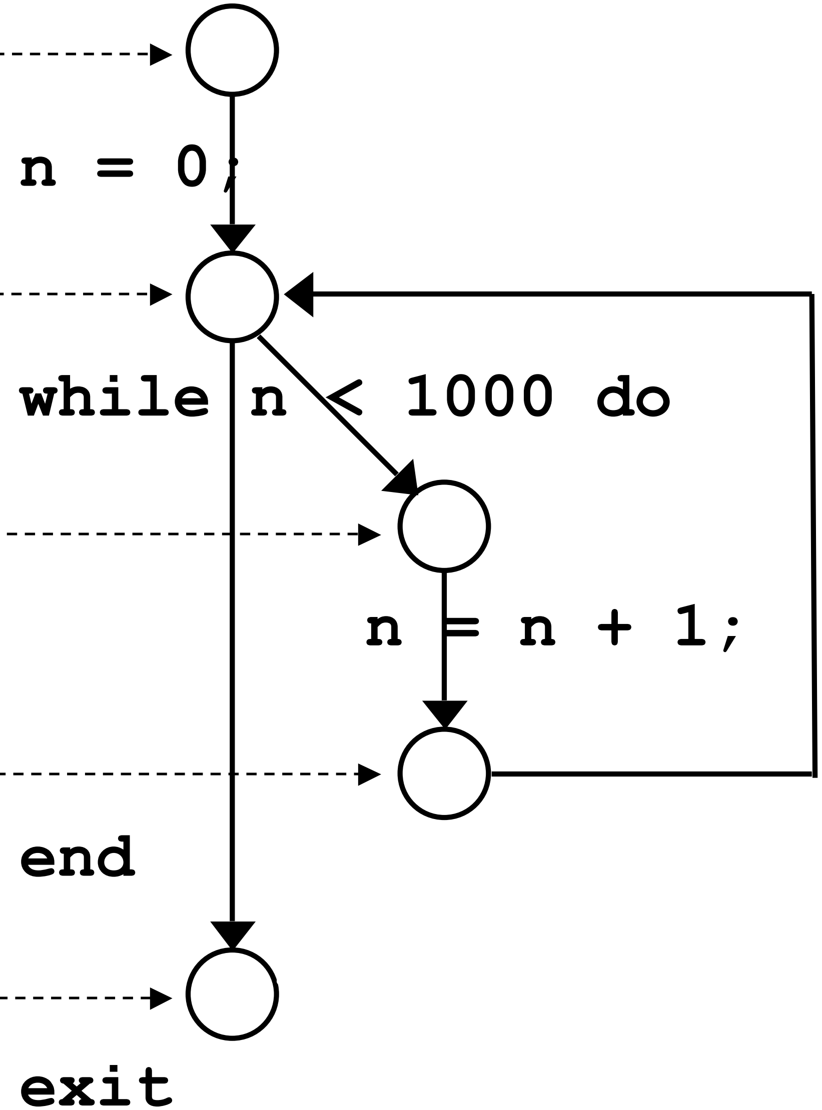
$$E = \{n \Rightarrow \Omega\}$$

$$E = [\![\mathbf{\texttt{n = 0}}]\!]\ E \cup E$$

$$E = E \cap\ ]-\infty, 999]$$

$$E = [\![\mathbf{\texttt{n = n + 1}}]\!]\ E$$

$$E = E \cap [1000, +\infty[$$

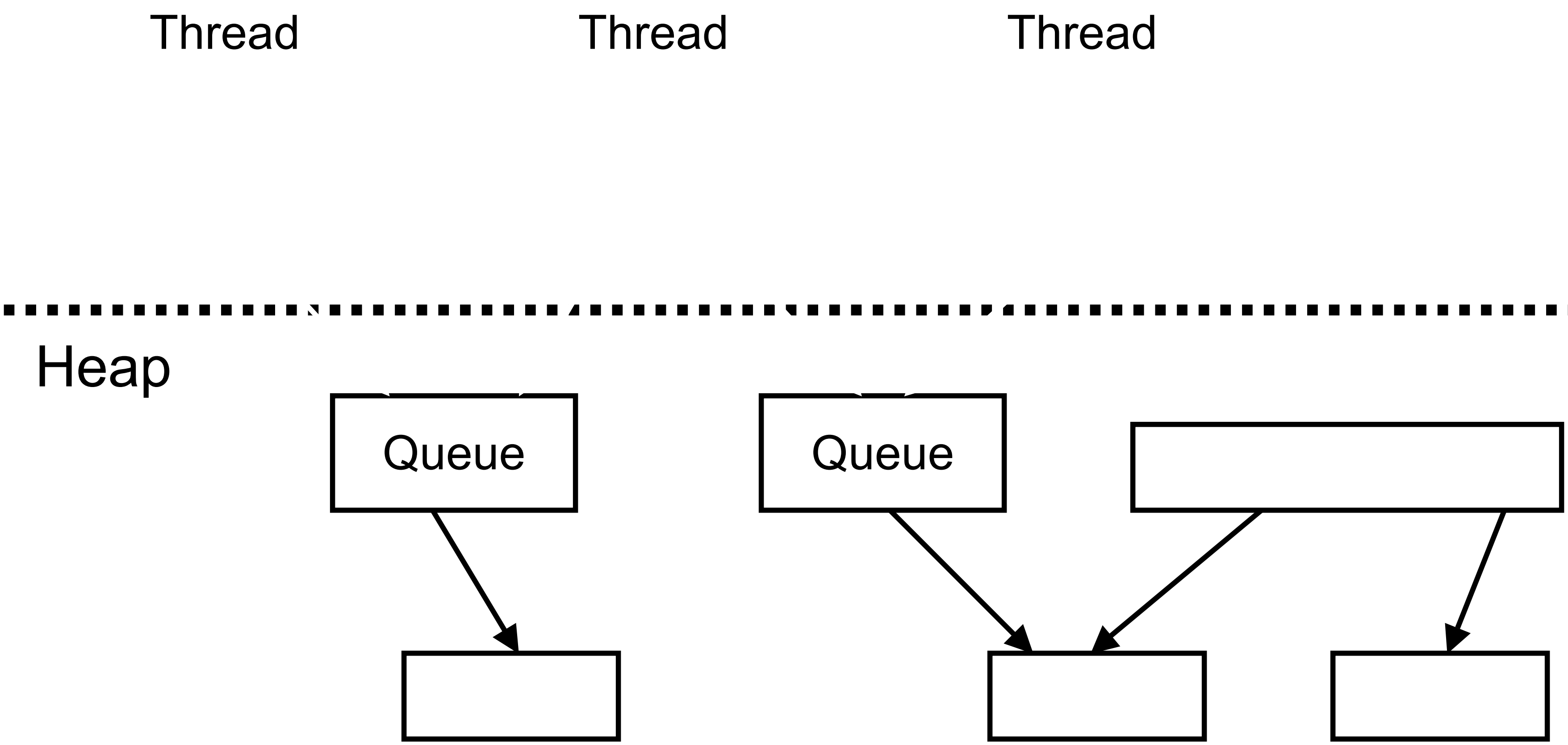


Simple Example

In effect, the analysis has automatically computed numerical invariants!

```
n = 0;

while n < 1000 do

         n = n + 1;

end

exit
```

MPF Flight Software Family



MPF Flight Software Family

```
assign (A, B, 10)          assign (&pS->f, &A[2], m)
```

```
assign (double *p, double *q, int n) {
  int i;
  for (i = 0; i < n; i++)
    p[i] = q[i];
}
```

The CGS Solution

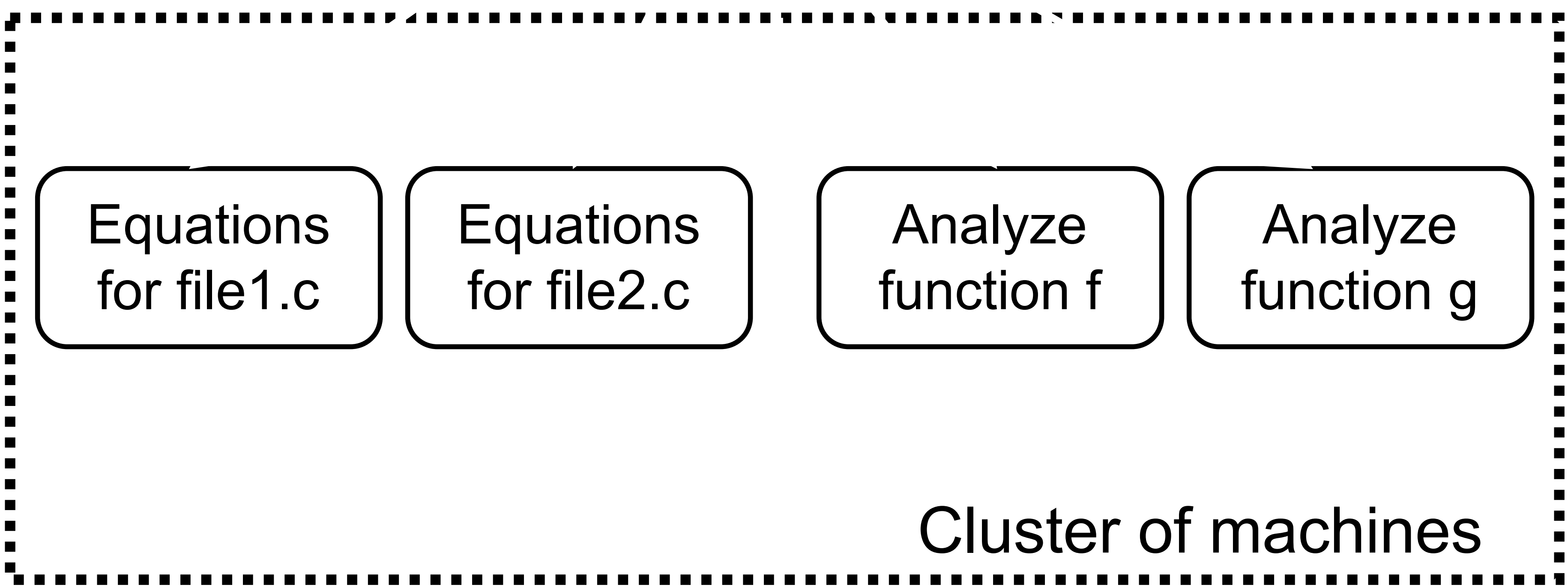
- Extensive representation using intervals
  - Some use of DBMs
  - Adaptive state variable clustering for scalability
- One level of context-sensitivity
- Computation of for speeding up the interprocedural propagation
- Parallel analyses over clusters of processors

Fast Context Sensitivity

- Context-sensitivity is required
- We can't afford performing 1000 fixpoint iterations with widening and narrowing for each function
- Compute a summary of the function using a relational numerical lattice

```
access(p[i],              )

access(q[i],              )
```

Implementation of CGS



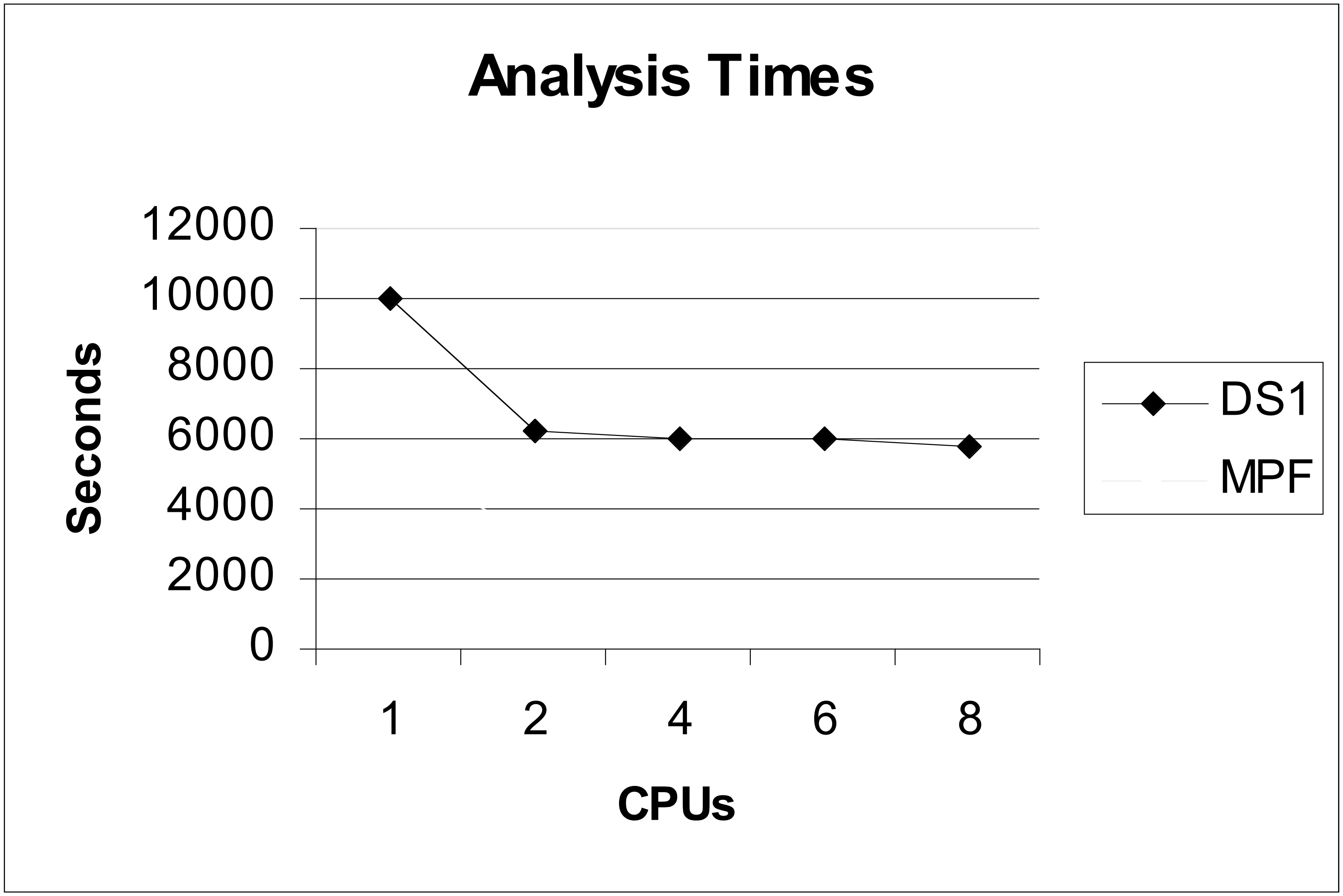
Working with a Database

- We use PostgreSQL
- Mutual exclusion problems are cared for by the database
- Simple interface using SQL queries

Parallel implementation

- We use the Parallel Virtual Machine (PVM)
- High-level interface for process creation and communication
- Allows heterogeneous implementation: currently a mix of C and OCaml

Effectiveness of Parallelization



The I/O Bottleneck

- The performance curve flattens: overhead of going through the network
- MER takes a bit less than 24 hours to analyze:
  - 70% of the time is spent in the interprocedural propagation
  - I/O times dominate (loading/unloading large tables)
- Under investigation: caching tables on machines of the cluster and using PVM communication mechanism (faster than concurrent database access)

Experimental Results

	Size (KLOC)	Max Size Analyzed	Precision	Analysis Time (hours)
MPF	140	140	80%	1.5
DS1	280	280	80%	2.5
MER	550	550	80%	20

C Global Surveyor

Conclusion

- NASA a besoin de meilleurs outils de vérification
- L’usage d’analyseurs statiques commerciaux s’est révélée décevante
  - Problèmes de passage à l’ échelle
  - Problèmes de précision
- Nous avons donc dévelopé notre propre outil d’analyse statique pour C
  - Passe à l’ échelle
  - Meilleurs temps d’analyse
  - Précision équivalente
- Prochaine étape: C++